

This exploded view diagram illustrates the various components of the HTC Vive VR headset. The parts are arranged to show their relative positions and assembly order. Key components include the main headset housing, the front-facing camera module, the internal display assembly, the external sensor base, the tracking sensors (labeled 'HTC VIVE'), the head strap, and various smaller mechanical parts like screws, gears, and a lens. The diagram is presented in a clean, white background, highlighting the intricate design of the device.

Gordon Wetzstein
Stanford University

Lecture 14

stanford.edu/class/ee267/

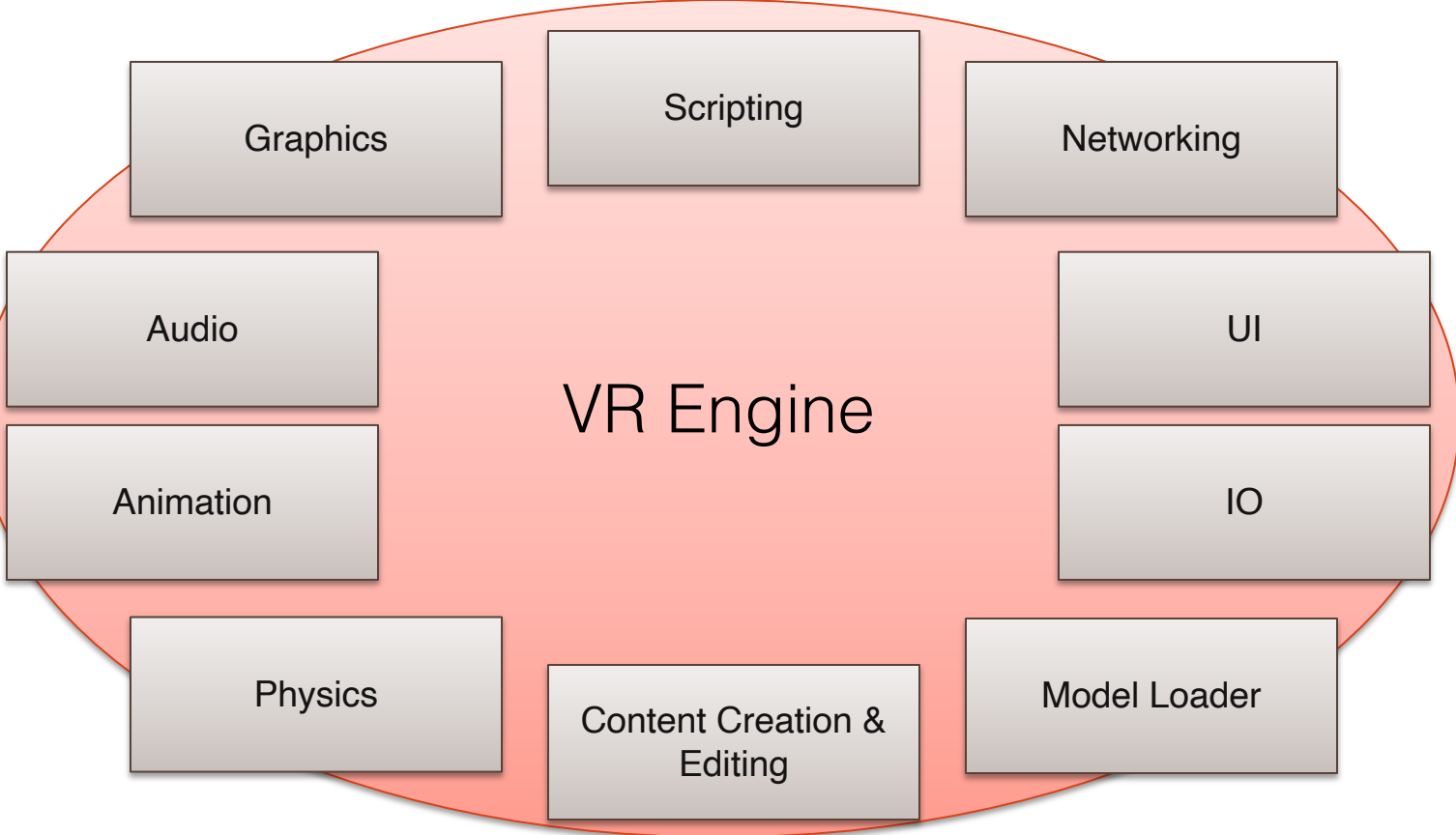


Graphics

The diagram consists of a large, light-red oval with a thin red border. Inside this oval, the text 'VR Engine (so far)' is centered. In the top-left corner of the oval, there is a gray rectangular box with a black border containing the word 'Graphics'. In the bottom-right corner of the oval, there is another gray rectangular box with a black border containing the letters 'IO'.

VR Engine
(so far)

IO



VR Engines - Audio

- middleware – between audio card and application (e.g. game)
- usually provides functionality for:
 - loading different types of sound files
 - mixing and mastering
 - 3D sound
 - occlusions, echoes, reverberation, ...

VR Engines - Audio

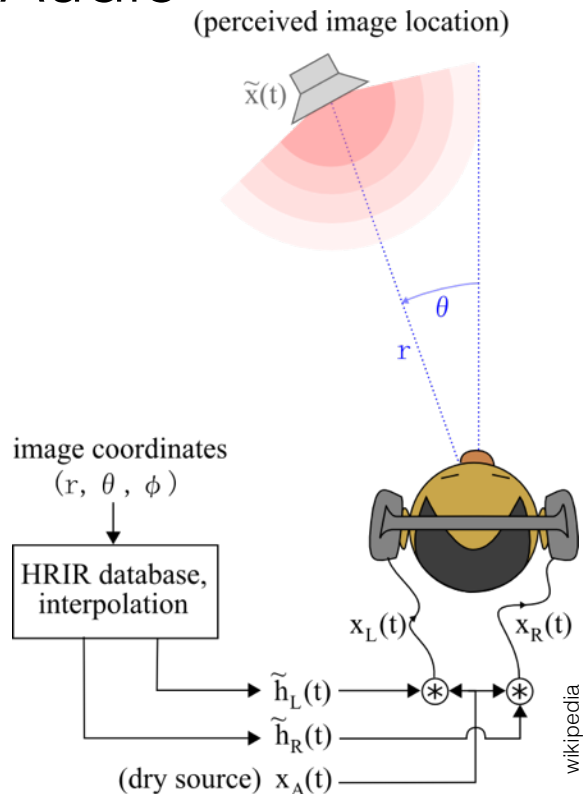
- examples:
 - FMOD - www.fmod.org
 - OpenAL - “OpenGL for sound”
 - SDL – provides basic functionality
 - ...

VR Engines – 3D Audio

- start with mono sound $x_A(t)$
- head-related impulse response (HRIR) model time delay and attenuation via convolution

$$\tilde{h}_L(t) \quad \tilde{h}_R(t)$$

- basically different temporal shift for each ear
- but HRIR also includes other effects created by shape of ear and other factors



VR Engines – 3D Audio

- sound, 3D sound, coupling sound and physics, accurate HRIR or head-related transfer function gets much more complicated
- Prof. Doug James in CS is working on physics & sound, check out his recent SCIEN talk if you're interested: "Physics-based Animation Sound: Progress and Challenges"

<https://talks.stanford.edu/doug-james-physics-based-animation-sound-progress-and-challenges/>

VR Engines - Physics

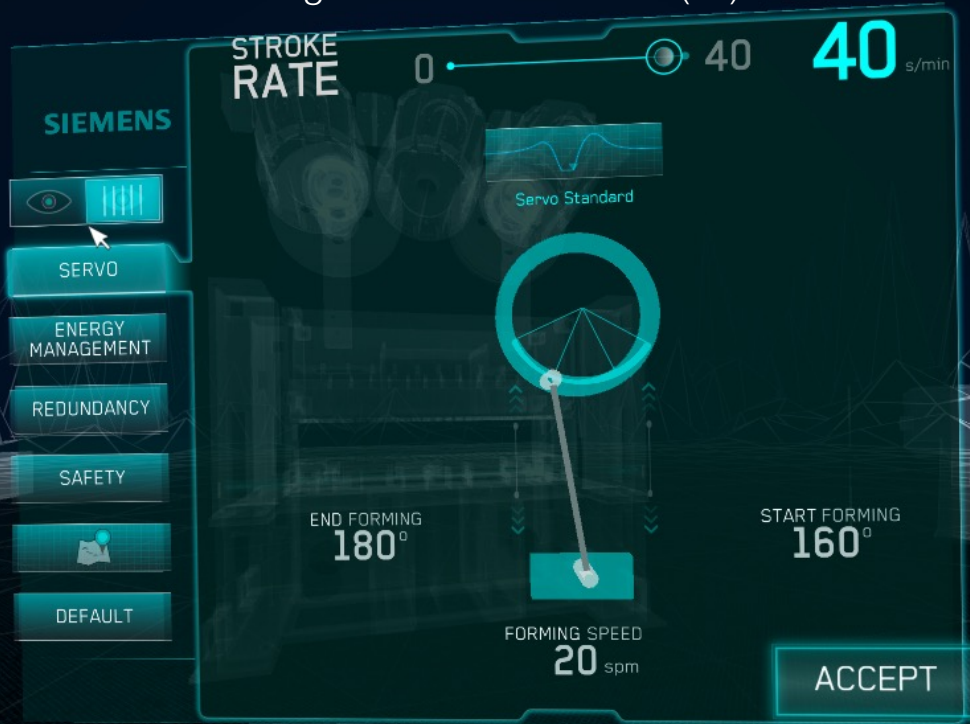
- framework to simulate:
 - rigid body dynamics (e.g. collision detection)
 - soft body dynamics (e.g. deformation, cloth, ...)
 - fluid dynamics (water, smoke, fire, ...)

VR Engines - Physics

- examples:
 - Open Dynamics Engine (<http://www.ode.org/>): free 😊 but limited to rigid body dynamics & collision
 - Bullet Physics (<http://bulletphysics.org/>): free 😊, rigid & soft body dynamics, widely used
 - havok (owned by Microsoft) – not free 😞 but widely used, real-time rigid body dynamics

Early Tests

VR Engines – User Interface (UI)



VR Engines – User Interface (UI)

- concept is straightforward: widgets, menus, buttons, checkboxes, ...
- types of UIs:
 - non-diegetic – lives in screen space (e.g. player status); doesn't work in VR (no screen space)
 - spatial UI – lives in the virtual world
 - diegetic – menus in world



VR Engines - IO

- support for interfaces: keyboard, mouse, 3D mouse, standard haptic devices, ...
- VR engine would provide functionality as well (e.g. Unity)

VR Engines – Content Creation

- 3D modeling programs / Computer-aided Design (CAD):
 - Maya (production)
 - 3ds Max (games)
 - Blender – free
 - SolidWorks – 3D printing & fabrication
 - Tinkercad: free & online

VR Engines – Content Creation

- what's involved?
 - conceptual design
 - 3D modeling
 - animation and/or simulation
 - scripting behavior and artificial intelligence of characters
 - testing
 - ... many different stages in application development ...

VR Engines - Scripting

- core engine is usually designed for performance – C++
- developing applications should be easy! the user almost never wants to touch the C++ source but needs flexibility
- provide a script-based interface to allow user to change anything they need for their application
 - create & manipulate objects
 - script behavior
 - change shaders (e.g. change camera or fragment shader art)

VR Engines - Networking

- manage low-level communication protocols (TCP/IP, UDP, ...)
- ensure that character states, graphics, sound, and everything else is synchronized
- connect to application that's running as client
- network updates, messages, ...

Popular VR/Game Engines

- Unity: cross-platform, Direct3D (Win), OpenGL (Mac & Linux), iOS & Android support, also came console APIs; personal license is free; seems to be the easiest to use so we'll use it for Lab 6 and HW 6
- Unreal: very popular, lots of awards, unreal engine 4 is free
- CryEngine: popular game engine, just announced support for VR; free for non-commercial use

Additional Information

- Unity game engine: <https://unity3d.com/>
- Unity tutorials: <https://unity3d.com/learn/tutorials>

Other Aspects of VR

Latency, Post-rendering Warp, Eye Tracking



Latency

- min acceptable: 20 ms
- interactive applications <20 ms
(say target is 5 ms)

The latency between the physical movement of a user's head and updated photons from a head mounted display reaching their eyes is one of the most critical factors in providing a high quality experience.

- John Carmack

Latency – where does it come from?

- IMU ~ 1 ms
- sensor fusion, data transfer
- rendering: depends on complexity of scene & GPU – a few ms
- data transfer again
- display: LCD ~ 60 Hz = 16 ms; OLED < 1 ms

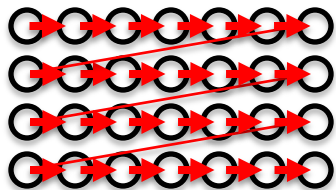
Latency – how bad is it really?

- example:
 - 16 ms (display) + 16 ms (rendering) + 4 ms (orientation tracking) = 36 ms latency total
 - head rotates at 60 degrees / sec (relatively slow)
 - 1Kx1K display over 100 degrees field of view
- in 36 ms, my head moved 1.92 deg ~ 19 pixels = size of thumb at arm's length! too much

Display Pixel Updates

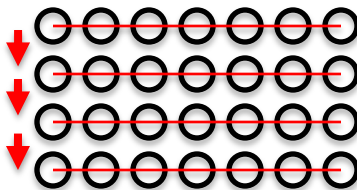
Raster Scan

(e.g. electron beam in CRT)



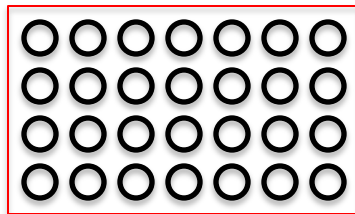
Rolling Update

(most LCDs)



Global Update

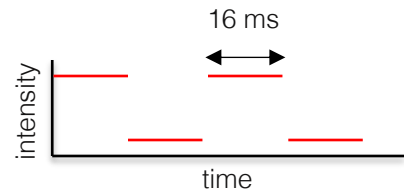
(some LCoS, DLP, other)



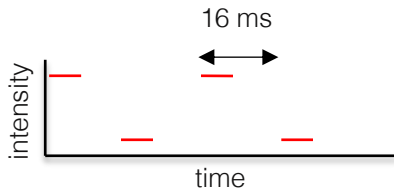
Display Pixel Switching - Persistence

- after the display pixel switched states, how long is it on?

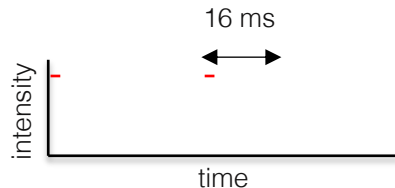
Full Persistence
(most LCDs)



Half Persistence
(strobe backlight)

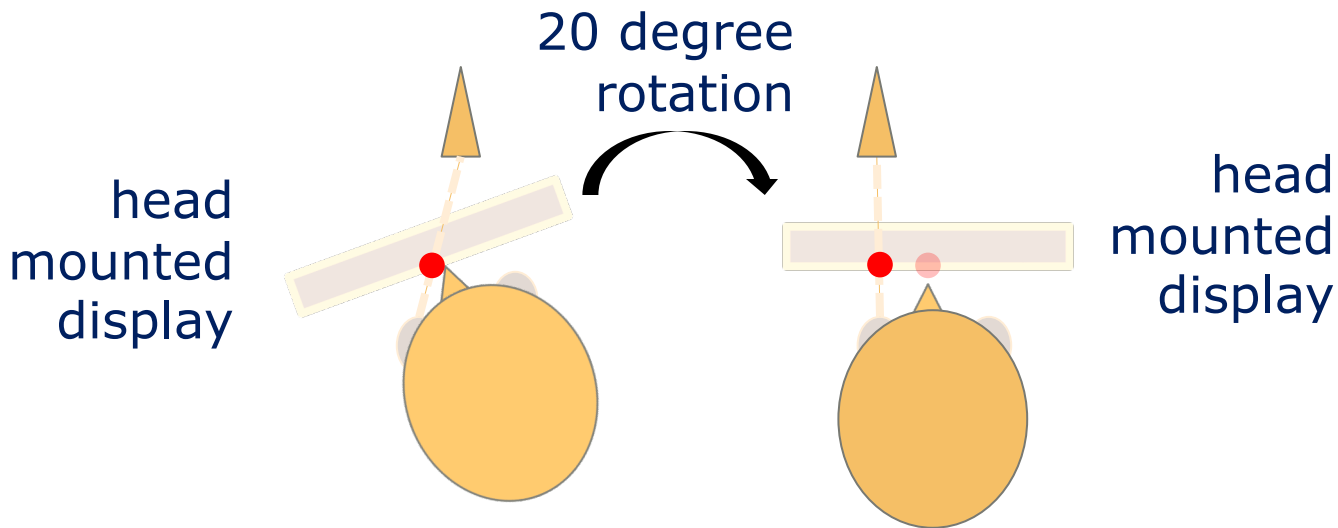


Low Persistence
(OLED, strobing)

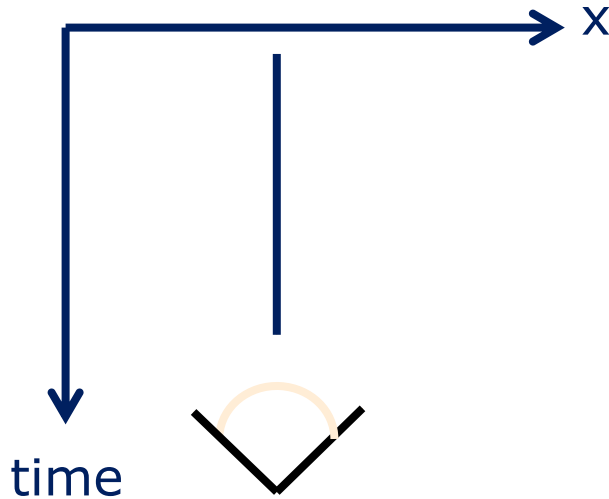


example: switch from white to black to white to black as fast as possible

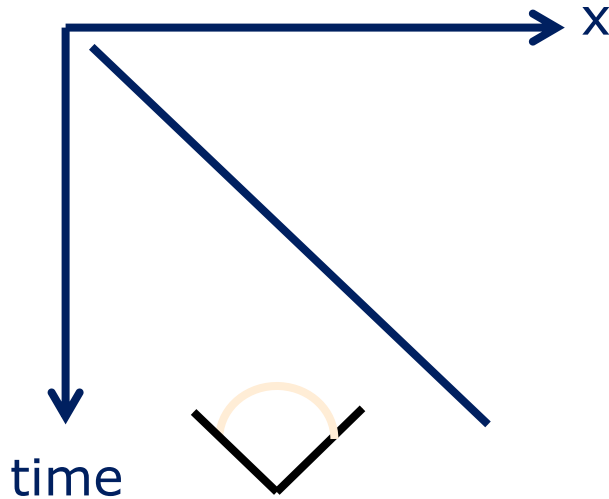
Rapid relative motion



Space-time diagrams (static)

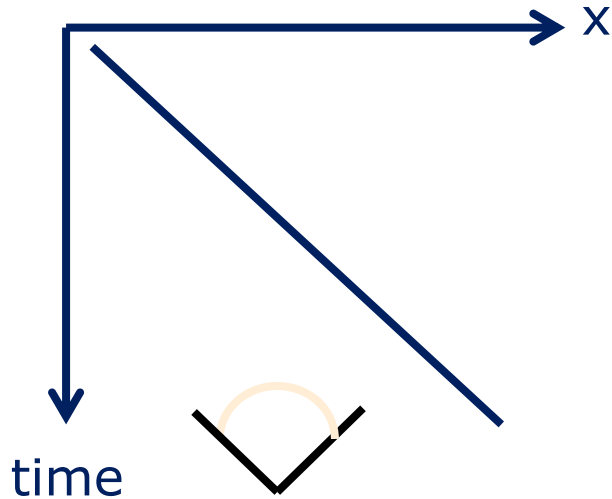


Spatial movement over time

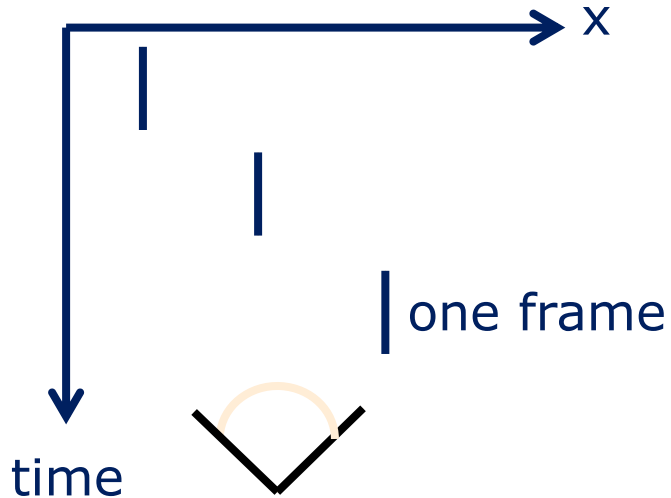




Spatial movement over time

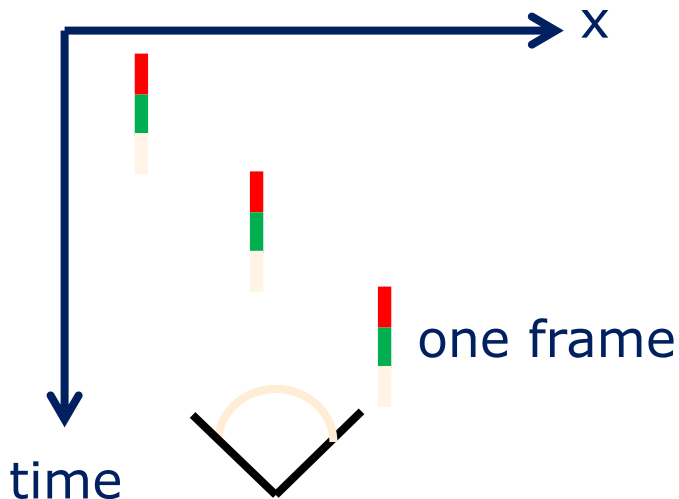


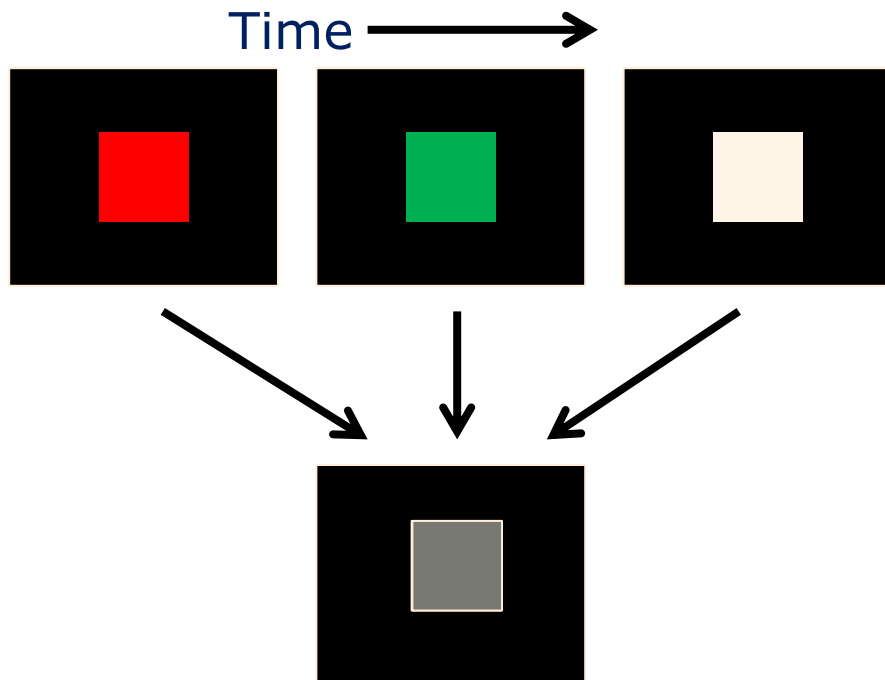
Pixel-based movement



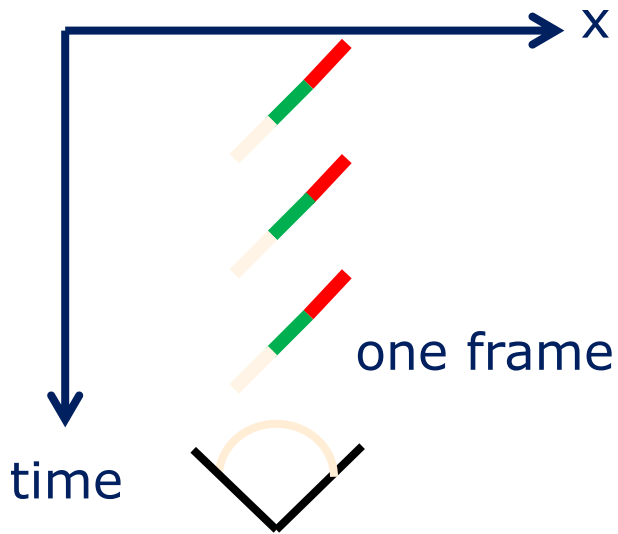


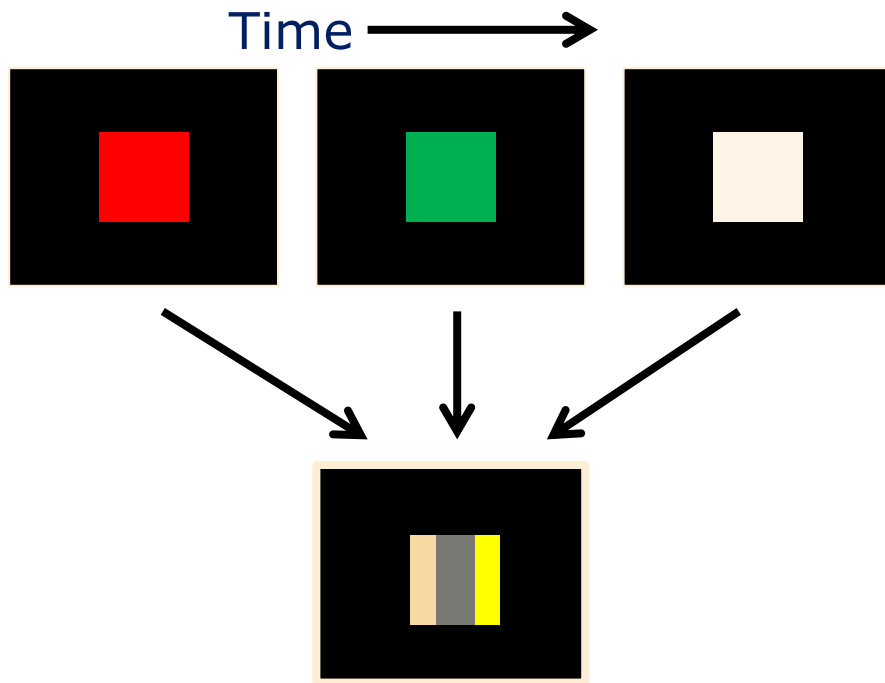
Sequential RGB display



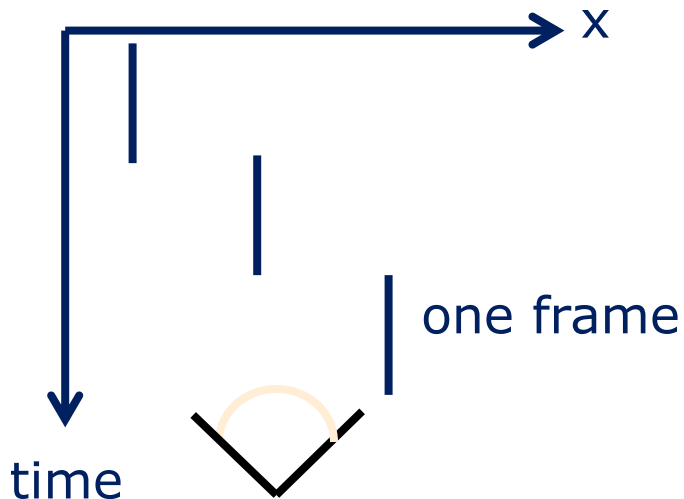


Sequential RGB with eyes moving

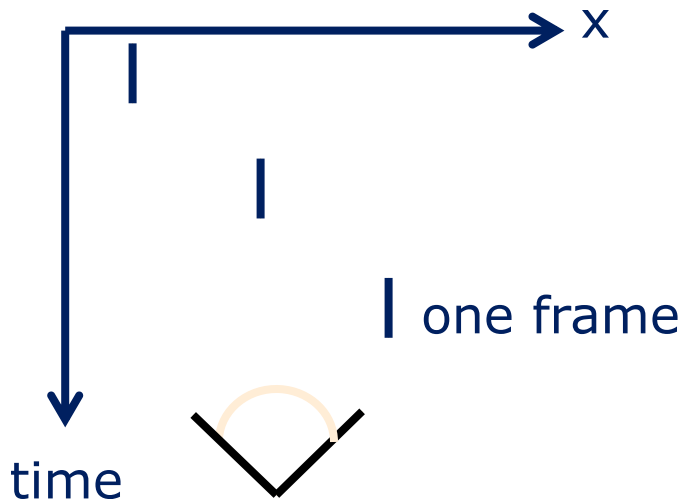




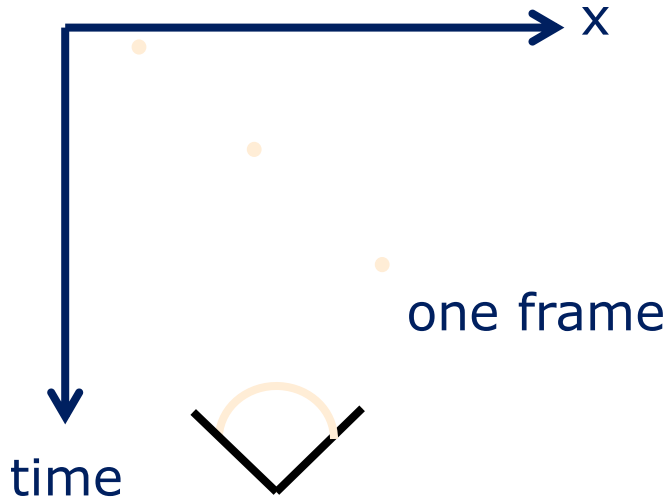
Full persistence



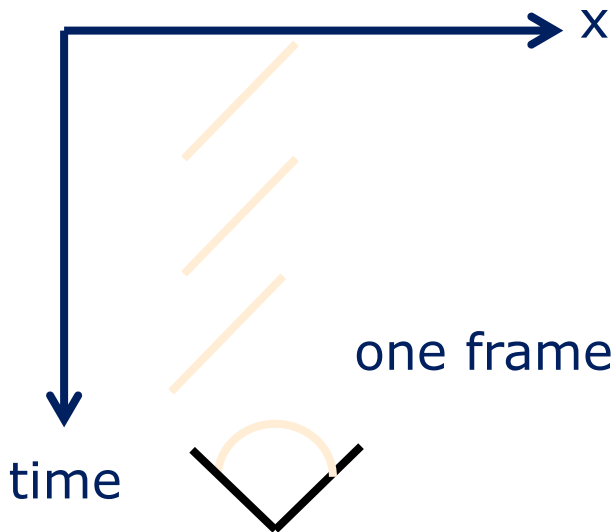
Half persistence



Zero persistence



Full persistence + head rotation





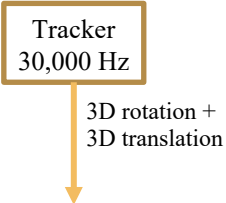
Post-rendering Image Warp

- also called time-warp by John Carmack
- minimize end-to-end latency
- original paper from Mark et al. 1997, also Darsa et al. 1997
- overview:
 1. get orientation from IMU, perhaps also position
 2. render scene into off-screen buffer (larger than screen)
 3. read latest orientation from IMU
 4. warp rendered image with latest orientation
- 2D image translation v 2D image warp v 3D image warp

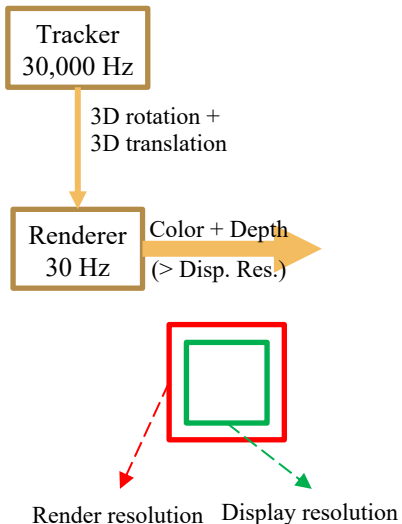
End-to-End Low-Latency Optical See-Through AR Pipeline

Tracker
30,000 Hz

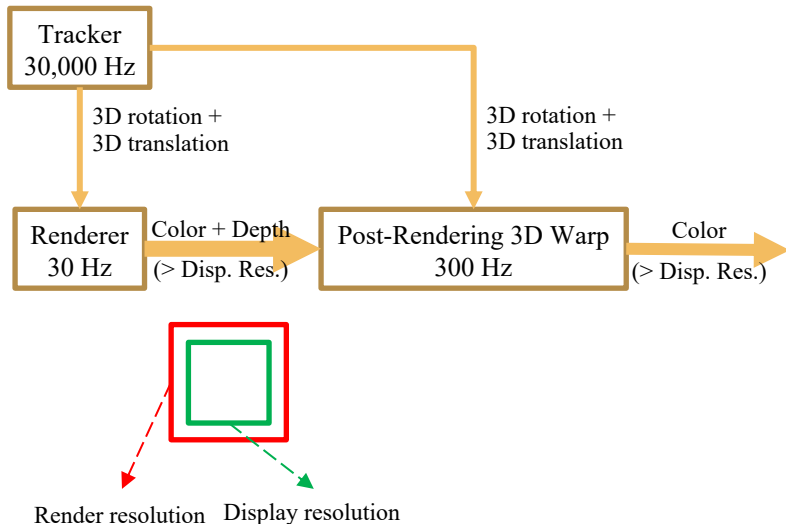
3D rotation +
3D translation



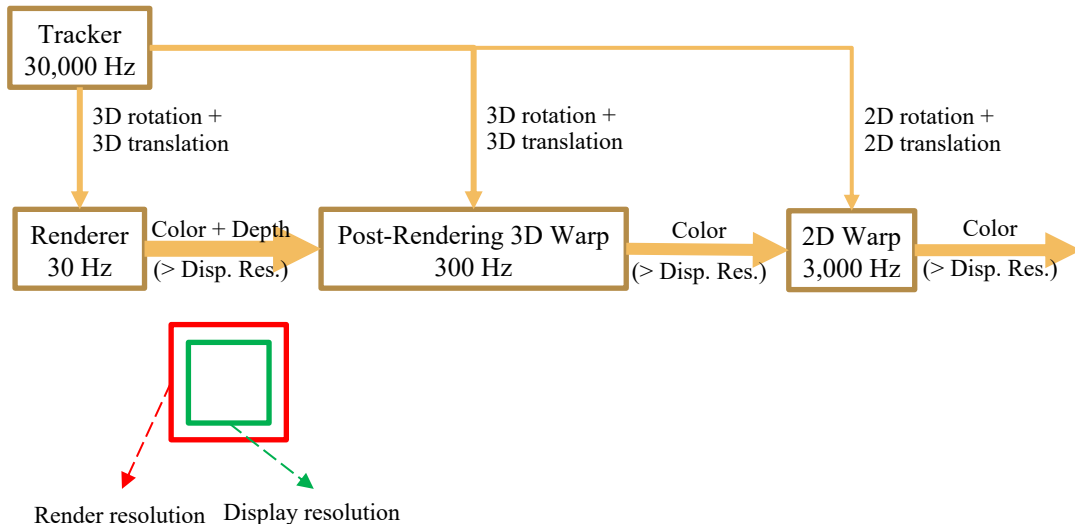
End-to-End Low-Latency Optical See-Through AR Pipeline



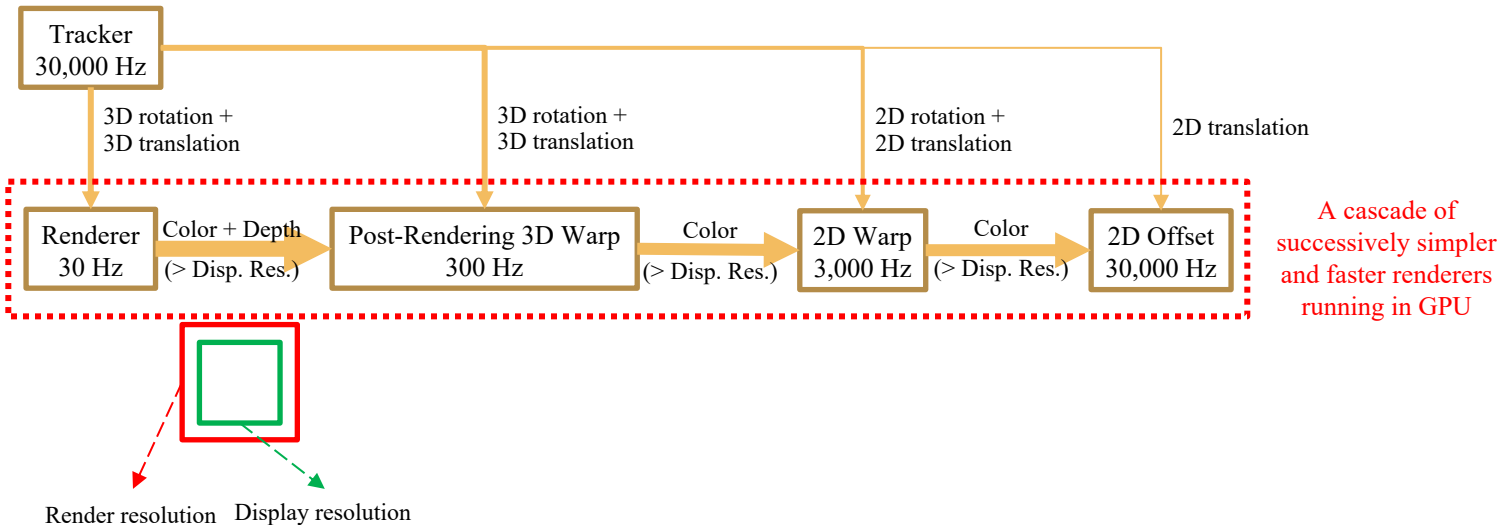
End-to-End Low-Latency Optical See-Through AR Pipeline



End-to-End Low-Latency Optical See-Through AR Pipeline



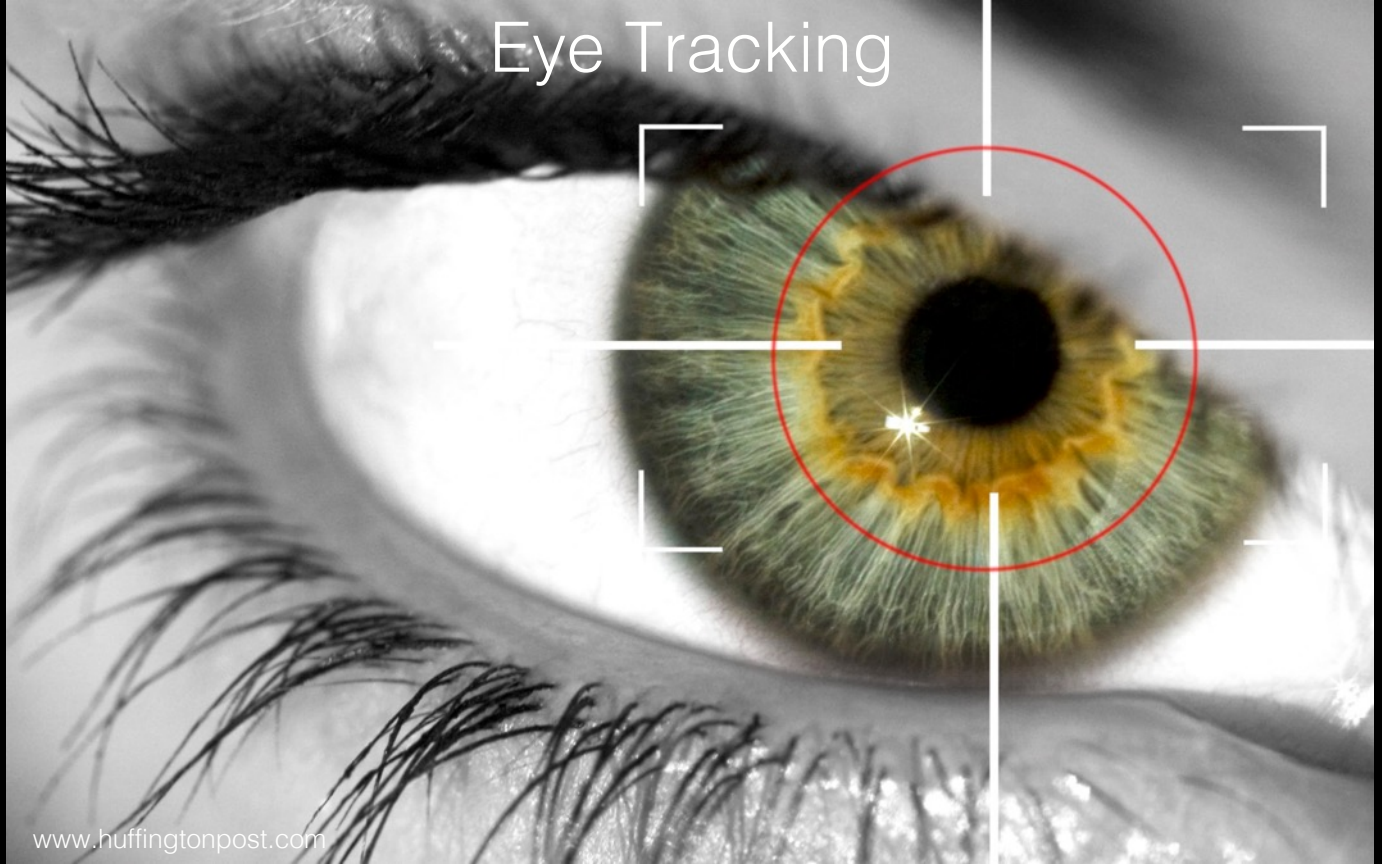
End-to-End Low-Latency Optical See-Through AR Pipeline



Summary: Latency, Persistence, etc.

- predictive tracking (e.g. LaValle ICRA 2014)
- post-rendering warp
- design and build really great hardware & algorithms
- use OLED displays or strobing backlights for low persistence
- design some type of a device to actually measure latency!

Eye Tracking



Eye Tracking

- necessary for gaze-contingency paradigm (foveated rendering, gaze-contingent rendering, gaze-contingent focus, ...)
- interaction
- eye contact
- ...

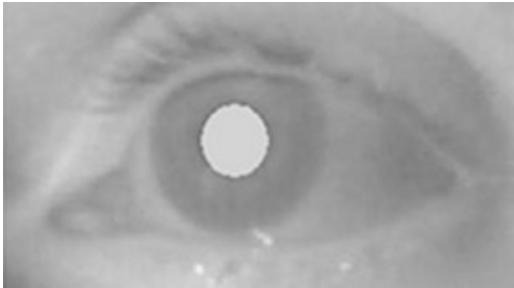
Eye Tracking

- many different techniques:
 - electro-oculography
 - contact lens tracking
 - video-oculography
 - pupil / corneal reflection tracking
 - dual Purkinje image

Eye Tracking

- some interesting properties one can exploit:
 - pupillary light reflex doesn't work in near infrared (IR)
 - red-eye effect with co-axial camera - light source
 - purkinje images for off-axis illumination

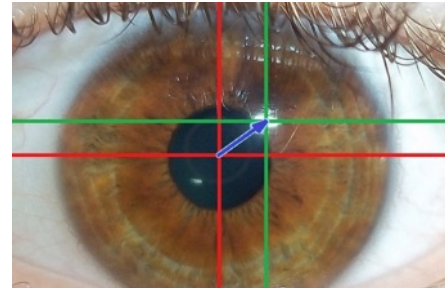
co-axial IR illumination



off-axis IR illumination



purkinje image

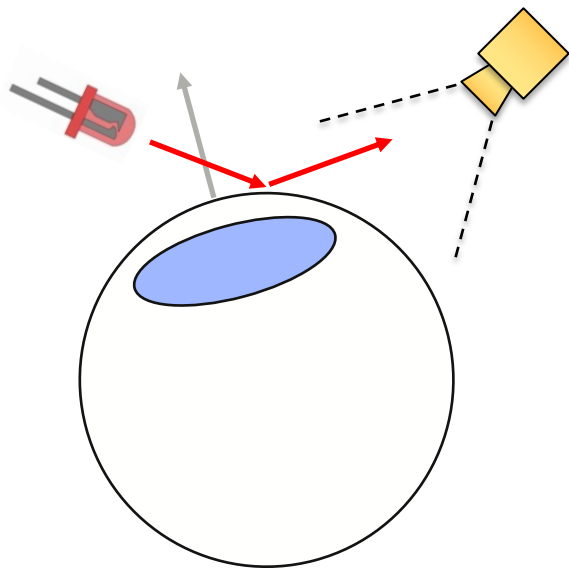
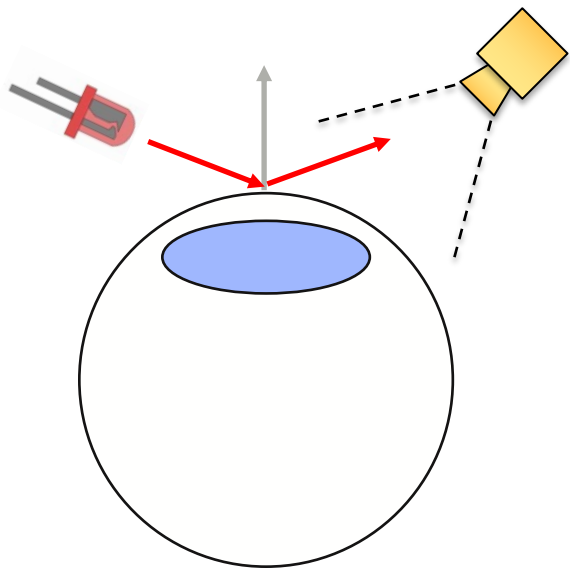


Eye Tracking – Pupil / Corneal Reflection



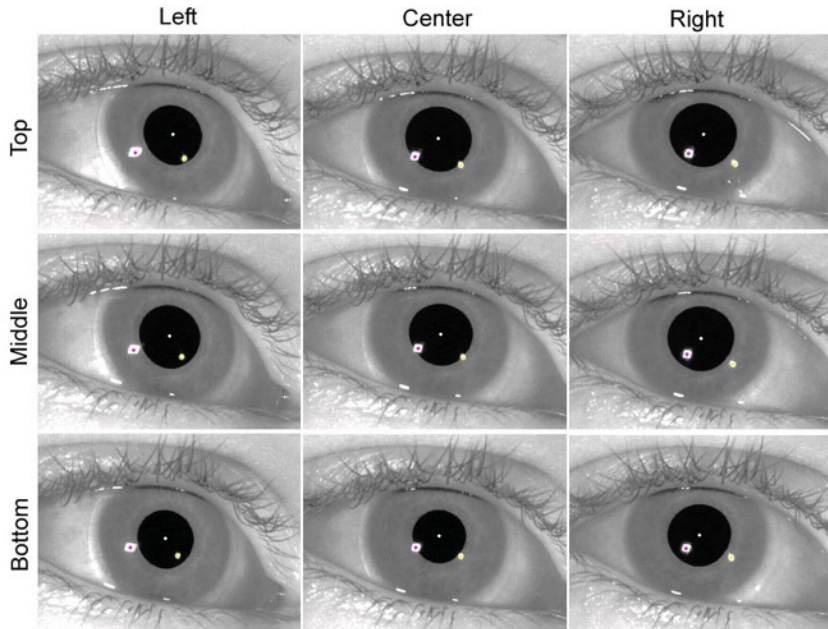
Eye Tracking – Pupil / Corneal Reflection

- corneal reflection stays constant, pupil center moves relative!



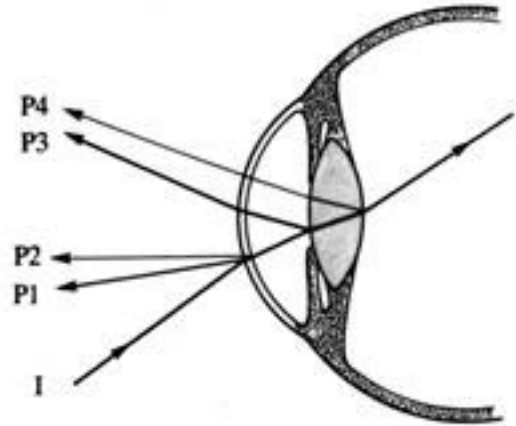
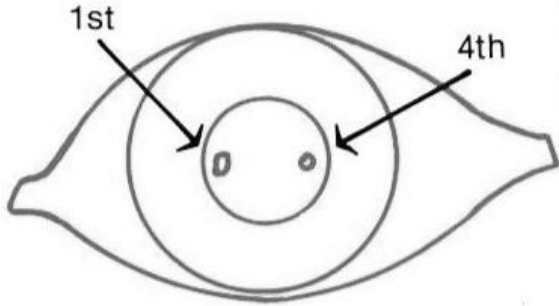
Eye Tracking – Pupil / Corneal Reflection

- corneal reflection stays constant, pupil center moves relative!



Eye Tracking – Dual Purkinje

- track relative location of Purkinje images



Eye Tracking

- where am I looking? what am I looking at?



References and Further Reading

- Google Project Tango: <https://developers.google.com/project-tango/>
- Post-rendering warp:
 - W. Mark, L. McMillan, G. Bishop "Post-Rendering Warping," Proc. Symposium on Interactive 3D Graphics 1997
 - L. Darsa, B. Costa, A. Varshneyz "Navigating Static Environments Using Image-Space Simplification and Morphing", 1997
 - John Carmack "Time Warp", 2013 (blogs)
- Latency:
 - F. Zheng, T. Whitted, A. Lastra, P. Lincoln, A. State, A. Maimonek, H. Fuchs "Minimizing Latency for Augmented Reality Displays: Frames Considered Harmful", ISMAR 2014